Micriμm

Empowering Embedded Systems

μC/OS-II

μC/OS-View

μC/TCP-IP

μC/TFTPs

μC/HTTPs

μC/FS

and

The **AT91SAM7X256**

(Using the AT91SAM7X-EK EVB)

Application Note

AN-9256

www.Micrium.com

Table Of Contents

1.00	Introduction	3
1.01	Source code access	4
1.02	Application Specific Considerations	5
2.00	The demo application	7
2.01	μC/OS-View	8
2.02	C-SPY Kernel Awareness Plug-in	10
3.00	Directories and Files	11
3.01	IAR Embedded Workbench	14
4.00	Example Code	15
4.01	Example Code, app.c	16
4.02	Example Code, app_cfg.h	21
4.03	Example Code, includes.h	21
4.04	Example Code, os_cfg.h	21
4.05	Ex1-OS-View-TCPIP-FS-HTTPs-TFTPs-FLASH.*	21
5.00	Board Support Package (BSP)	22
5.01	Board Support Package, bsp*.*	22
5.02	Board Support Package, net_bsp.c	25
6.00	Demo Application	28
6.01	HTTP Server demo	28
6.02	TFTP Server	30
6.03	Ping	31
	References	32
	Contacts	32

1.00 Introduction

This document shows example code for using μC/OS-II, μC/OS-View, μC/TCP-IP, μC/DHCPc, μC/FS, μC/TFTPs and μC/HTTPs on a Atmel AT91SAM7x-EK evaluation board which is based on the Atmel AT91SAM7x256 CPU (ARM7). The SAM7 board is shown in Figure 1-1.

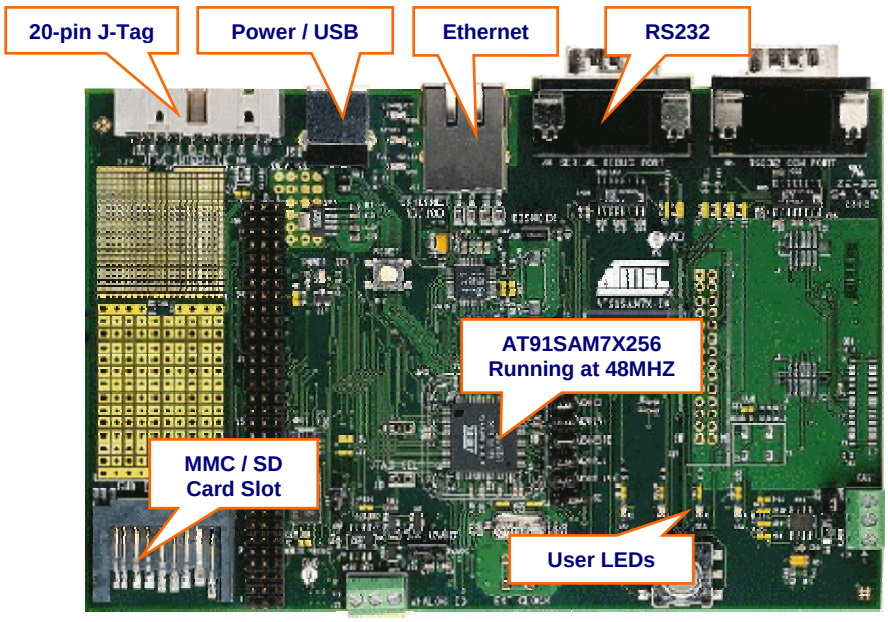


Figure 1-1, Atmel AT91SAM7X-EK EVB

1.01 Source code access

This application note describes a demonstration application using µC/OS-II, µC/OS-View, µC/TCP-IP, µC/FS, µC/TFTPs and µC/HTTPs. This application is delivered in binary version only. The source code can be made available to customers who would have secured licenses for µC/OS-II, µC/OS-View, µC/TCP-IP, µC/FS, µC/TFTPs and µC/HTTPs. (Please contact Micrium (sales@micrium.com).

This application note is based on µC/OS-II Version 2.81. You can get access to earlier version of µC/OS-II as it is included on a CD-ROM with the book written by Jean Labrosse (http://www.cmpbooks.com/product/1057).

Micrium's policy in regards to getting access to source code is as follows:

> µC/OS and µC/OS-II source and object code can be used by accredited Colleges and Universities without requiring a license, as long as there is no commercial application involved. In other words, no licensing is required if µC/OS and µC/OS-II is used for educational use.
>
> You need to obtain an 'Object Code Distribution License' to embed µC/OS or µC/OS-II in a product that is sold with the intent to make a profit or if the product is not used for education or 'peaceful' research.

For all the µC/ products other than µC/OS-II, you need to obtain an 'Object Code Distribution License' to get access to the source code.

1.02 Application Specific Considerations

The Application Note described herein is intended to be loaded into the target as a Flash memory application. RAM operation of the OS is possible, but not with all of the modules described below. For situations where the OS and application code require less than 64KB of memory, the symbol 'RAM_REMAPPED' can be defined to allow the writing of the interrupt vectors into RAM during initialization. Although this symbol can be defined anywhere, it is used in bsp.c and should either be defined there, or in the setup options for your compiler.

In addition to the above, it should be noted that the AT91SAM7X-EK EVB may experience inconsistent and undesirable operation when running the EMAC in RMII mode. In accordance to a suggestion made by Atmel, we have chosen to operate the EMAC in MII mode. The NIC driver however supports both modes of operation and can be easily toggled between both modes by defining the symbol 'RMII' when desiring RMII mode of operation. Not defining this symbol implies the MII operation mode. We have defined this in the compiler settings for the project. The current definition is 'COMMENT_RMII' which is to indicate that 'RMII' is not defined, but could be by removing the word COMMENT and the underscore that follows.

Note: Not all versions of the AT91SAM7X-EK EVB have all of the parts necessary to run in MII mode. According to Atmel, your board can be run in MII mode if the following criteria are met, or can be modified such that:

Crystal Y3 = 25MHz
C46 and C47 = 22pF

Cut bridges S23 and S25
Solder bridges S24 and S26

Next, μC/OS-View is implemented using the RS232 Communication port, as opposed to the DEBUG Communication port. Both ports are clearly labeled on the surface of the EVB just in front of their respective connectors. If you intend to run μC/OS-View, please ensure that your cable is connected to the correct port and that the baud rate is set for 38400, 8N1.

Also, the AT91SAM7X-EK comes equipped with a MMC/SD card slot (labeled Data Flash Card on the EVB). For code portability, we have chosen to write a 'bit-banging' hardware layer using general purpose IO pins for μC/FS. The performance under this mode of operation depends on the rate of the timer used by μC/OS-View. Since both μC/OS-View and the μC/FS MMC/SD hardware layer share some of the same requirements, they have been configured to use the same system timer. If desired, the timer being used can be changed directly in mmc_X_hw.c by changing the macro responsible for choosing the system timer within SPI_Delay(). The new macro name should be defined in app_cfg.h. A good name for this macro may be 'FS_TIMER_SEL'.

Furthermore, μC/TCP-IP uses the AT91SAM7X256 internal EMAC interface for network connectivity. Please note that this application note hard codes a MAC address in app.c which cannot be re-used. Please take special care to ensure that duplicate MAC addresses do not appear on the same Ethernet network as a result of using this application note. Micrium does not supply MAC addresses for use with μC/TCP-IP and low level Ethernet device drivers. It is up to the end user to purchase a group of MAC addresses from the IEEE. Please see

http://standards.ieee.org/regauth/oui/index.shtml for more information about requesting an OUI or company ID.

Note: The device IP Address is configured within app.c. You may hard code a new IP address here, or alternatively, contact Micrium with regard to using µC/DHCP in your product.

Next, this AT91SAM7X256 example is limited by the amount of on chip RAM available. Although the entire project when compiled uses less than 64KB, the remaining memory, about 15KB, has been dedicated to the heap for use by µC/HTTPs. However, µC/HTTPs currently requires the heap to be twice the size of the largest file that you wish to transmit. As a result, this example can only transmit a file via HTTP that is less than 7.5KB in size. Having external memory would greatly increase this limit.

It is important to understand that this example does not use the switch on the MMC/SD card slot for card detection. As a result, µC/FS assumes the presence of the media during initialization. If the media is not present during initialization, the example will not recognize the card if inserted or removed and reinserted at a later time. If you intend to test the file system in this example, please ensure that media is inserted before µC/FS initialization takes place and remains inserted throughout testing.

Finally, not all versions of IAR EWARM have support for the AT91SAM7X256 in the processor selection menu. The configuration files to enable support for this processor are included in the application note under the directory below. If you open this example and go to your compiler settings only to find that the processor selection is not correct, please copy these files into their respective IAR directories. This can be determined by matching the file extensions with other files of the same extension and placing the new files in the same place.

\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\IAR EWARM SAM7X256 Configuration Files

2.00 The demo application

Figure 2-1 shows a block diagram that depicts the relationship between the components needed to run μC/OS-II, μC/OS-View, μC/TCP-IP, μC/TFTPs, μC/FS, and μC/HTTPs in a typical embedded target application.

The demo application implements several tasks under μC/OS-II. μC/OS-II internally creates three tasks: the idle task, the statistics task and the timer task. μC/OS-View creates one task, μC/TCP-IP creates two tasks, μC/HTTPs and μC/TFTPs create one task each, and finally, the core demo application creates one task, the start task.

μC/HTTPs and μC/TFTPs also assume the presence of a File System which allows you to read files from some mass storage device (RAM disk, on-board Flash, True IDE, SD/MMC, etc.). Both μC/HTTPs and μC/TFTPs assume the API of Micrium's μC/FS so, if you have a different file system, it would be easier to write a small API adaptation layer to simulate μC/FS than to change μC/HTTPs and μC/TFTPs.

You may notice that μC/TFTPs is needed to build the example application. This module has been used in the example for convenience. Loading example HTML files on to the SD/MMC card for use with μC/HTTPs may be done via TFTP using the IP address configured in `app.c`. Alternatively, the SD/MMC card can be loaded before initializing the target via a PC if you have an SD/MMC card reader.

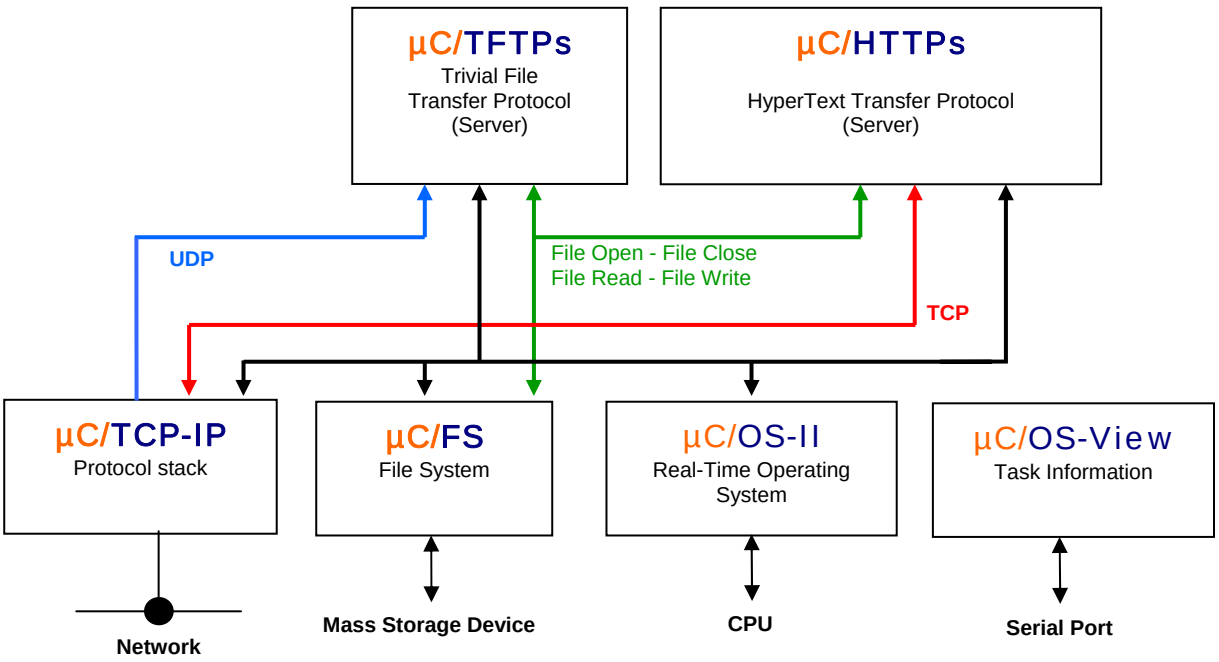


Figure 2-1, Relationship between the different modules.

The demonstration application is very simple. It initializes all of the above modules and toggles on board LEDs in a predefined pattern. Once the target has been fully initialized, the user can interact with the target by viewing web pages stored on the MMC/SD card, or by using TFTP to read or write new files to the target.

μC/OS-View can be used to view operating system information such as the number of tasks, processor utilization, and the status of each task (delayed, running, stack usage etc…) while the target is running. As mentioned above, μC/OS-View is configured to use the communication port labeled 'RS232 COM Port' and is configured for 38,400 baud by default.

2.01 μC/OS-View

The application code described in this application note allows you to connect a Windows-based PC to your target and display run-time information about your target in a Window as shown in Figure 2-2. This is done via an add-on module called μC/OS-View.

Note that you can 'disable' μC/OS-View by removing the μC/OS-View files from the build and setting `OS_VIEW_MODULE` to `0` in `os_cfg.h`. You would need to do this is you didn't purchase μC/OS-View from Micriμm.

μC/OS-View is a combination of a Microsoft Windows application program and code that resides in your target system (in this case, the AT91SAM7X-EK). The Windows application connects with your system via an RS-232C serial port. The Windows application allows you to '**View**' the status of your tasks which are managed by μC/OS-II.

μC/OS-View allows you to view the following information from a μC/OS-II based product:

The address of the TCB of each task (up to 63 tasks)
The name of each task (up to 253 tasks)
The status (Ready, delayed, waiting on event) of each task
The number of ticks remaining for a timeout or if a task is delayed
The amount of stack space used and left for each task
The percentage of CPU time each task relative to all the tasks
The number of times each task has been 'switched-in'
The execution profile of each task
More.

μC/OS-View also allows you to send commands to your target and allow your target to reply back and display information in the applications 'terminal window'. A terminal callback function is present in `app.c` and allows the developer to capture the received characters.

μC/OS-View is licensed on a per-developer basis. In other words, you are allowed to install μC/OS-View on multiple PCs as long as the PC is used by the same developer. If multiple developers are using μC/OS-View then each needs to obtain their own copy. Contact Micriμm for pricing information.

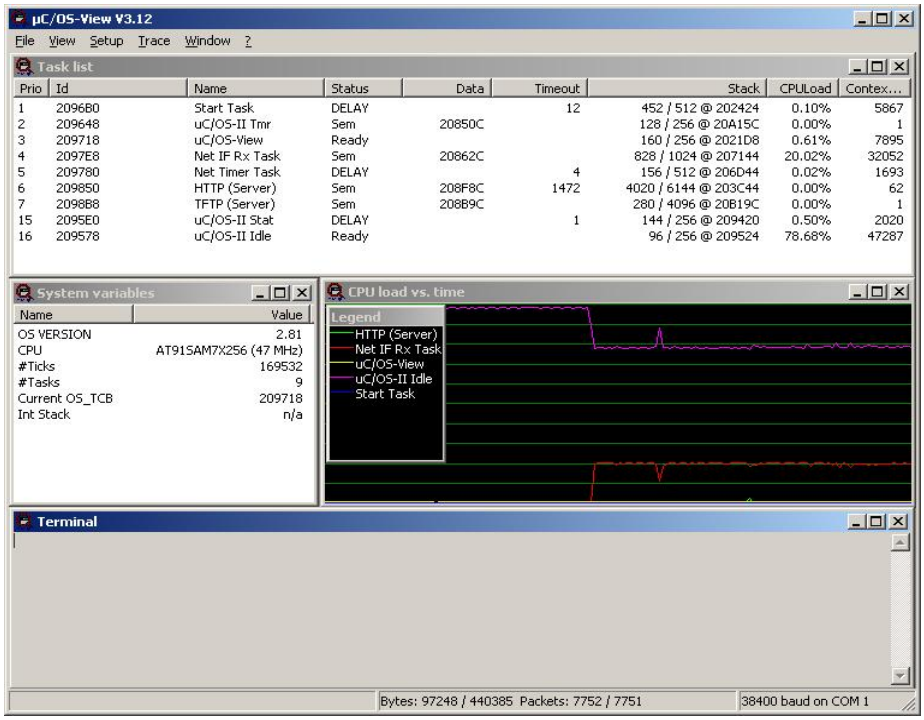

μC/OS-View V3.12
File View Setup Trace Window ?
Task list
Prio Id Name Status Data Timeout Stack CPULoad Contex...
1 2096B0 Start Task DELAY 12 452 / 512 @ 202424 0.10% 5867
2 209648 uC/OS-II Tmr Sem 20850C 128 / 256 @ 20A15C 0.00% 1
3 209718 uC/OS-View Ready 160 / 256 @ 2021D8 0.61% 7895
4 2097E8 Net IF Rx Task Sem 20862C 828 / 1024 @ 207144 20.02% 32052
5 209780 Net Timer Task DELAY 4 156 / 512 @ 206D44 0.02% 1693
6 209850 HTTP (Server) Sem 208F8C 1472 4020 / 6144 @ 203C44 0.00% 62
7 2098B8 TFTP (Server) Sem 208B9C 280 / 4096 @ 20B19C 0.00% 1
15 2095E0 uC/OS-II Stat DELAY 1 144 / 256 @ 209420 0.50% 2020
16 209578 uC/OS-II Idle Ready 96 / 256 @ 209524 78.68% 47287
System variables
Name Value
OS VERSION 2.81
CPU AT91SAM7X256 (47 MHz)
#Ticks 169532
#Tasks 9
Current OS_TCB 209718
Int Stack n/a
CPU load vs. time
Legend
HTTP (Server)
Net IF Rx Task
uC/OS-View
uC/OS-II Idle
Start Task
Terminal
Bytes: 97248 / 440385 Packets: 7752 / 7751
38400 baud on COM 1

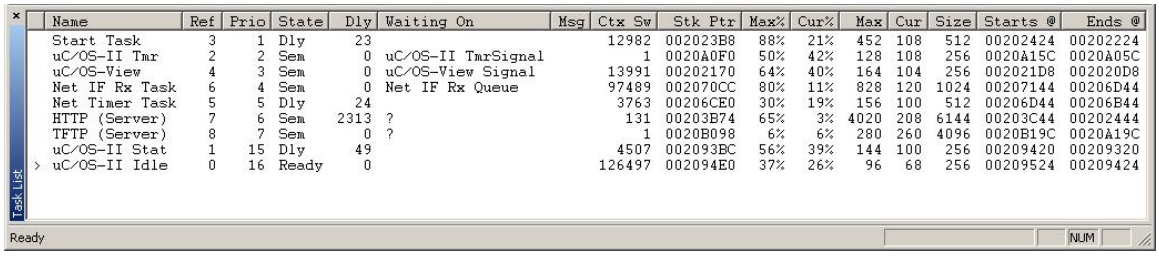
Figure 2-2, μC/OS-View Windows' 'Viewer'

2.02 C-SPY Kernel Awareness Plug-in

The IAR Embedded Workbench is available with Micrium's μC/OS-II Kernel Awareness Plug-In which allows you to examine μC/OS-II kernel objects in tabular format when running the IAR C-Spy debugger.

Figure 2-3 shows all the tasks created in the AT91SAM7x256 example. Each task can be assigned a name, you can also see where the current stack pointer for each task is pointing to, how much stack space each task is using and more.

The Kernel Awareness Plug-In also provides useful information about other μC/OS-II objects (semaphore list, mailbox list, queue list, etc.).



Name	Ref	Prio	State	Dly	Waiting On	Msg	Ctx Sw	Stk Ptr	Max%	Cur%	Max	Cur	Size	Starts @	Ends @
Start Task	3	1	Dly	23			12982	002023B8	88%	21%	452	108	512	00202424	00202224
uC/OS-II Tmr	2	2	Sem	0	uC/OS-II TmrSignal		1	0020A0F0	50%	42%	128	108	256	0020A15C	0020A05C
uC/OS-View	4	3	Sem	0	uC/OS-View Signal		13991	00202170	64%	40%	164	104	256	002021D8	002020D8
Net IF Rx Task	6	4	Sem	0	Net IF Rx Queue		97489	002070CC	80%	11%	828	120	1024	00207144	00206D44
Net Timer Task	5	5	Dly	24			3763	00206CE0	30%	19%	156	100	512	00206D44	00206B44
HTTP (Server)	7	6	Sem	2313	?		131	00203B74	65%	3%	4020	208	6144	00203C44	00202444
TFTP (Server)	8	7	Sem	0	?		1	0020B098	6%	6%	280	260	4096	0020B19C	0020A19C
uC/OS-II Stat	1	15	Dly	49			4507	002093BC	56%	39%	144	100	256	00209420	00209320
uC/OS-II Idle	0	16	Ready	0			126497	002094E0	37%	26%	96	68	256	00209524	00209424

Figure 2-3, μC/OS-II Kernel Awareness in C-Spy, Task List

3.00 Directories and Files

The code and documentation of the port are placed in a directory structure according to "AN-2002, μC/OS-II Directory Structure". Specifically, the files are placed in the following directories:

μC/OS-II:

\Micrium\Software\uCOS-II\Source

This directory contains the processor independent code for μC/OS-II. The version used was 2.81.

\Micrium\Software\uCOS-II\Ports\ARM

This directory is the main directory for ARM7 and ARM9 ports.

\Micrium\Software\uCOS-II\Ports\ARM\Generic

This directory contains the 'generic' port of μC/OS-II which works in either ARM or Thumb mode. Download **AN-1014** from the Micrium web site for details.

\Micrium\Software\uCOS-II\Ports\ARM\Generic\IAR

This directory contains the standard processor specific files for a μC/OS-II port assuming the IAR tool chain. In fact, these files could easily be modified to work with other tool chains. However, you would place the modified files in a different directory. Specifically, this directory contains the following files:

```
os_cpu.h
os_cpu_a.s
os_cpu_c.c
os_dbg_c
```

`os_dbg.c` is included to provide additional information to Kernel Aware debuggers like IAR's C-Spy.

As mentioned above, you can either compile the code for ARM or Thumb mode. The port is fully described in application note AN-1014 which is available from the Micrium web site. The files are:

```
AN-1014.PDF
AN-1014-PPT.PDF
```

Application Code:

\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\Ex-1-OS-View-TCPIP-FS-HTTPs-TFTP

This directory is the directory that contains the source code for this example running on the AT91SAM7X-EK evaluation board. This directory contains:

```
app.c
app_cfg.h
fs_conf.h
includes.h
net_cfg.h
os_cfg.h
Ex1-OS-View-TCPIP-FS-HTTPs-TFTP.*
```

`app.c` contains the example code,

`app_cfg.h` contains application specific configuration information such as task priorities and stack sizes.

`fs_conf.h` contains configuration parameters for μC/FS,

`includes.h` contains a master include file used by the application,

`net_conf.h` contains μC/TCP-IP configuration parameters and,

`os_cfg.h` is the μC/OS-II configuration file.

`Ex1-OS-View-FS-TCPIP-DHCPc-HTTPs-TFTP.*` are the IAR Embedded Workbench project files.

\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\BSP

This directory contains the Board Support Package for the AT91SAM7X256-EK Evaluation Board from Atmel. This directory contains:

```
bsp.c
bsp.h
flash at912sam7-no-remap.xcl
FlashAT91SAM7X256.mac
ioat91sam7x256.h
mmc_X_hw.c
net_bsp.c
net_bsp.h
cstartup.s79
```

`bsp.c` contains source code necessary for configuring AT91SAM7X256 board specific inputs and outputs as well as the interrupt service routines and their initialization functions. However, for flash based applications, the ISR vectors are initialized in `cstartup.s79`.

`bsp.h` contains function prototypes for some of the functions within `bsp.c`

`ioat91sam7x256.h` is an IAR provided header file for the AT91SAM7X256. The register and bit definitions within this file are used throughout this example.

`flash at912sam7-no-remap.xcl` is a linker file used to specify the memory address that the example should be linked to.

`FlashAT91SAM7X256.mac` is a macro file used by EWARM to performs tasks such as configuring the AT91SAM7X256 memory map prior to loading the application code into the target.

`mmc_X_hw.c` is the a low level GPIO hardware access routines for use with the MMC / SD card and µC/FS.

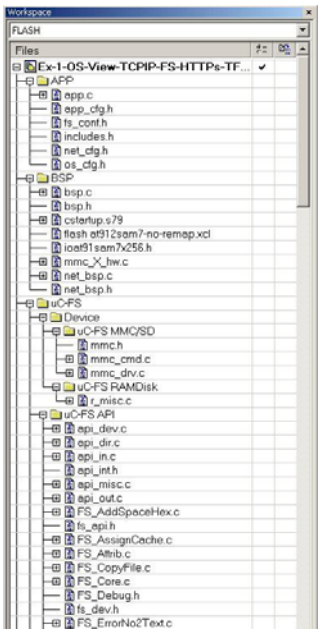
Finally, `net_bsp.c` and `net_bsp.h` contain low level hardware access routines which make up the AT91SAM7X256 EMAC network driver.

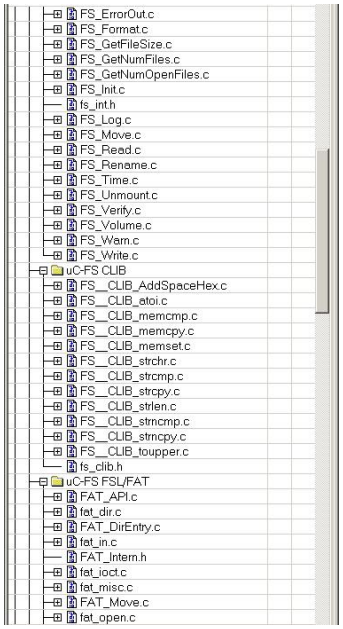
`\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\DOC`

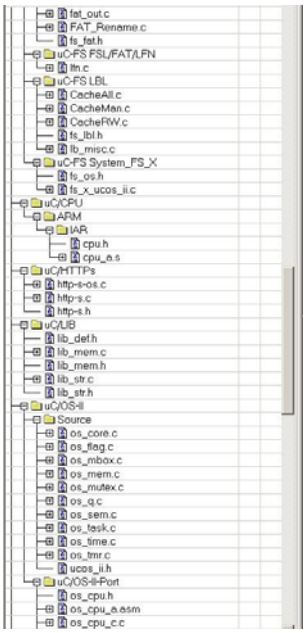
This directory contains the documentation for the AT91SAM7X256 example.

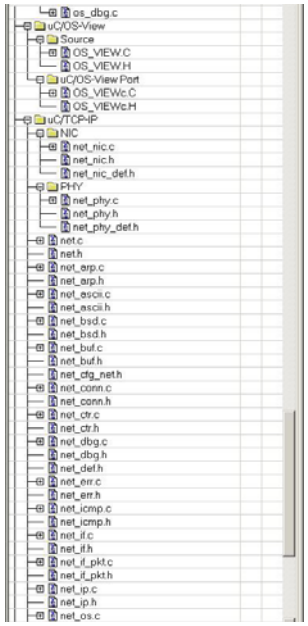
3.01 IAR Embedded Workbench

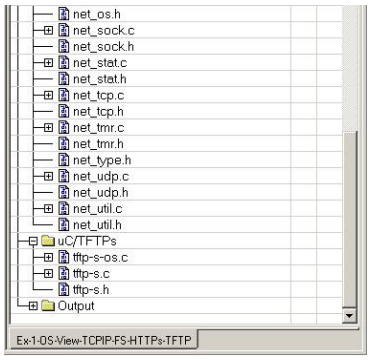
We used the IAR Embedded Workbench (EW) V4.30a to test this application note. You can of course use Micriμm products with other tools. In EWARM, the project source tree is read from left to right, and then top row to bottom row. As you can see, this project contains a lot of files.











4.00 Example Code

As mentioned in the previous section, the example code for this board is found in the following directory and will be briefly described:

\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\Ex-1-OS-View-TCPIP-FS-HTTPs-TFTP

The example code works either in ARM or Thumb mode. In fact, you can simply select ARM or Thumb ***Processor Mode*** (see Figure 4-1) and 'rebuild all' the code and it will run just as well. However, we run this code in THUMB mode to reduce its code size.

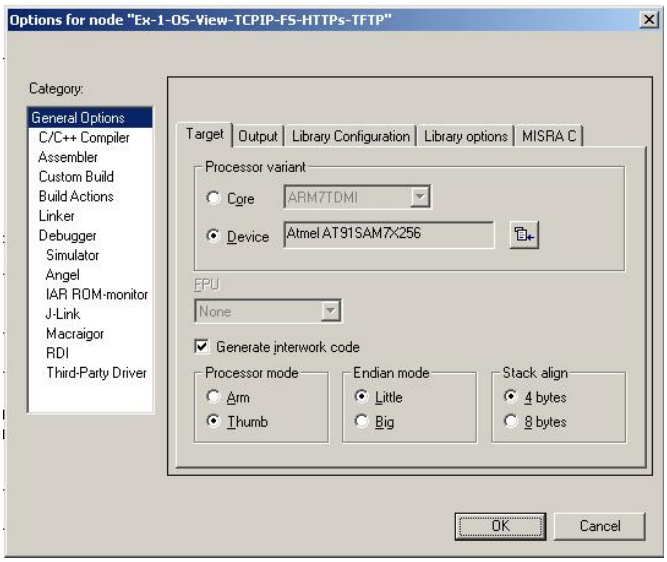


Figure 4-1, Building either ARM or Thumb mode code

4.01 Example Code, app.c

app.c demonstrate some of the capabilities of μC/OS-II, μC/OS-View, μC/TCP-IP, μC/DHCPc, μC/HTTPs, μC/FS and the μC/OS-II Kernel Awareness plug-in for C-Spy (all available from Micriμm). Only the critical functions within app.c are described in details. The others are briefly described at the end of the section.

Listing 4-1, main()

```
void  main (void)                                                    (1)
{
    INT8U  err;


    BSP_IntDisAll();                                                 (2)

    OSInit();                                                        (3)

    OSTaskCreateExt(AppTaskStart,                                    (4)
                    (void *)0,
                    (OS_STK *)&AppTaskStartStk[APP_TASK_START_STK_SIZE - 1],
                    APP_TASK_START_PRIO,
                    APP_TASK_START_PRIO,
                    (OS_STK *)&AppTaskStartStk[0],
                    APP_TASK_START_STK_SIZE,
                    (void *)0,
                    OS_TASK_OPT_STK_CHK | OS_TASK_OPT_STK_CLR);

#if OS_TASK_NAME_SIZE > 13
    OSTaskNameSet(APP_TASK_START_PRIO, " Start Task", &err);       (5)
#endif

    OSStart();                                                       (6)
}
```

L4-1(1) As with most C applications, the code starts in main().

L4-1(2) We start off by calling a BSP (Board Support Package) function (see bsp.c) that will disable all interrupts. We do this to ensure that initialization doesn't get interrupted in case we do a 'warm restart'.

L4-1(3) As will all μC/OS-II applications, you need to call OSInit() before creating any task or other kernel objects.

L4-1(4) We then create at least one task (in this case we used OSTaskCreateExt() to specify additional information about your task to μC/OS-II). It turns out that μC/OS-II creates one and possibly three tasks in OSInit(). As a minimum, μC/OS-II creates an idle task (OS_TaskIdle() which is internal to μC/OS-II), OS_TaskStat() (if you set OS_TASK_STAT_EN to 1 in OS_CFG.H) and, a timer task (if you set OS_TMR_EN to 1 in OS_CFG.H).

L4-1(5) As of V2.6x, you can now name μC/OS-II tasks (and other kernel objects) and be able to display task names at run-time or, with a debugger. In this case, we name our first task as well as the two internal μC/OS-II tasks.

L4-1(6) We finally start μC/OS-II by calling `OSStart()`. μC/OS-II will then start executing `AppStartTask()` since that's the highest priority task created.

Listing 4-2, AppTaskStart()

```
static  void  AppTaskStart (void *p_arg)
{
    (void)p_arg;                                                    (1)

    BSP_Init();                                                     (2)

#if OS_TASK_STAT_EN > 0
    OSStatInit();                                                   (3)
#endif

#if OS_VIEW_MODULE > 0
    AppInit_OSView();                                               (4)
#endif

#if uC_TCPIP_MODULE > 0
    AppInit_TCPIP();                                                (5)
#endif

#if uC_FS_MODULE > 0
    AppInit_FS();                                                   (6)
    FSTest();                                                       (7)
#endif

    LED_Off(0);                                                     (8)

    while (TRUE) {                                                  (9)
        for (j = 0; j < 4; j++) {
            for (i = 1; i <= 4; i++) {
                LED_On(i);                                         (10)
                OSTimeDlyHMSM(0, 0, 0, 25);                        (11)
                LED_Off(i);                                        (12)
                OSTimeDlyHMSM(0, 0, 0, 25);
            }

            for (i = 3; i >= 2; i--) {
                LED_On(i);
                OSTimeDlyHMSM(0, 0, 0, 25);
                LED_Off(i);
                OSTimeDlyHMSM(0, 0, 0, 25);
            }
        }

        for (i = 0; i < 4; i++) {
            LED_On(0);
            OSTimeDlyHMSM(0, 0, 0, 50);
            LED_Off(0);
            OSTimeDlyHMSM(0, 0, 0, 50);
        }

        if (OSCPUUsageMax < OSCPUUsage) {
            OSCPUUsageMax = OSCPUUsage;                            (13)
        }
    }
}
```

L4-2(1) `(void)p_arg` prevents a common compiler warning which says that `p_arg` is not referenced anywhere in the code.

L4-2(2) `BSP_Init()` is called to initialize the BSP (Board Support Package) – the I/Os, the tick interrupt, and so on. `BSP_Init()` will be discussed in the next section.

L4-2(3) `OSStatInit()` initializes the statistics task which is responsible for counting context switches and so fourth. `OS_TASK_STAT_EN` in `os_cfg.h` is set to `1`.

L4-2(4) If `OS_VIEW_MODULE > 0`, then call `AppInit_OSView()` to initialize μC/OS-View.

L4-2(5) If `uC_TCPIP_MODULE > 0`, then call `AppInit_TCPIP()` to initialize μC/TCP-IP.

L4-2(6) If `uC_FS_MODULE > 0`, then call `AppInit_FS()` to initialize μC/FS.

L4-2(7) If `uC_FS_MODULE > 0 (app_cfg.h)`, then perform a file system test.

L4-2(8) Turn all user LEDs off.

L4-2(9) As with ALL task managed by μC/OS-II, the task must either enter an infinite loop 'waiting' for some event to occur or terminate itself.

L4-2(10) `LED_On()` allows on board LEDs to be switched on. This function has been designed to take values between 0 and 4 that correspond to each of the onboard LEDs, 0 referring to all LEDs.

L4-2(11) As with ALL task managed by μC/OS-II, the task must either enter an infinite loop 'waiting' for some event to occur or terminate itself. We decided to simply have the task delay itself for 500 mS each time the LED's are switched on or off.

L4-2(12) `LED_Off()` allows on board LEDs to be switched off. This function has been designed to take values between 0 and 4 that correspond to each of the onboard LEDs, 0 referring to all LEDs.

L4-2(13) For the web page application, check the `OSCPUUsage` and if its higher than the highest previously recorded value, save the value as the new maximum CPU usage.

Listing 4-3, AppInit_OSView()

```
static void AppInit_OSView (void)
{
    OSView_Init(38400);                                          (1)
    OSView_TerminalRxSetCallback(AppTerminalRx);                 (2)
    OSView_RxIntEn();                                            (3)
}
```

L4-3(1) `OSViewInit()` is called to initialize μC/OS-View if `OS_VIEW_MODULE` is defined in `os_cfg.h`. The baud rate is passed as a parameter and can be 9600, 19200, 38400 (default), 57600 or 115200.

L4-3(2) `OSView_TerminalRxSetCallback()` sets the call back function for the μC/OS-View terminal window. When a character is entered, it can be processed in the specified routine. In this case, `AppTerminalRx()`.

L4-3(3) `OSView_RxIntEn()` enables the UART1 interrupt source so that we may begin processing μC/OS-View RS232 packets.

Listing 4-4, AppInit_TCPIP()

```
static void AppInit_TCPIP (void)
{

#if AT91SAM7X256_EMAC_CFG_MAC_ADDR_SEL == EMAC_CFG_MAC_ADDR_SEL_CFG (1)
    NetIF_MAC_Addr [0] = 0x00;
    NetIF_MAC_Addr [1] = 0x50;
    NetIF_MAC_Addr [2] = 0xC2;
    NetIF_MAC_Addr [3] = 0x25;
    NetIF_MAC_Addr [4] = 0x60;
    NetIF_MAC_Addr [5] = 0x01;
#endif

    err = Net_Init();                                                   (2)

    ip      = NetASCII_Str_to_IP("192.168.0.60",  &err);                (3)
    msk     = NetASCII_Str_to_IP("255.255.255.0", &err);
    gateway = NetASCII_Str_to_IP("192.168.0.1",   &err);
    err     = NetIP_CfgAddrThisHost(ip, msk);
    err     = NetIP_CfgAddrDfltGateway(gateway);

#if uC_TCPIP_HTTPs_MODULE > 0
    HTTPs_Init();                                                       (4)
#endif

#if uC_TFTP_MODULE > 0
    TFTPs_Init(OS_TICKS_PER_SEC);                                       (5)
#endif
}
```

L4-4(1) If `AT91SAM7X256_EMAC_CFG_MAC_ADDR_SEL` is defined as `EMAC_CFG_MAC_ADDR_SEL_CFG`, that is to say, the MAC address is user defined in software, then this is where the user specifies the device MAC address.

L4-4(2) `Net_Init()`is called to initialize µC/TCP-IP stack.

L4-4(3) The user should specify an IP, Netmask, and Gateway address for µC/TCP-IP. After converting the dotted decimal notation to 32 bit values, a call to both `NetIP_CfgAddrThisHost()` and `NetIP_CfgAddrDfltGateway()` is made in order to configure µC/TCP-IP to use the specified addresses.

L4-4(4) If the µC/HTTPs module is present, start the µC/HTTP server.

L4-4(5) If the µC/TFTPs module is present, start the µC/TFTP server.

Listing 4-5, AppInit_FS()

```
static void AppInit_FS (void)
{
    FS_Init();                                                          (1)
#if FS_USE_RAMDISK_DRIVER > 0                                           (2)
    FS_IoCtl("ram:", FS_CMD_FORMAT_MEDIA, FS_MEDIA_RAM_512KB, 0);
#endif
}
```

L4-5(1) Start µC/FS

L4-5(2) If the RAM Disk driver is enabled, format the RAM Disk with 512KB sectors. Note that you need external memory in order to use the RAM drive since the AT91SAM7x256 only has 64Kbytes of internal RAM.

Listing 4-6, HTTPs_ValReq ()

```
CPU_BOOLEAN  HTTPs_ValReq (CPU_CHAR *Variable, CPU_CHAR **Val, CPU_INT32U MaxSize)
{
                                                                    (1)
…
    if (Str_Cmp(Variable, "OS_VERSION") == 0) {                     (2)
        ver = (CPU_FP32)OS_VERSION / 100;                           (3)
        Str_FmtNbr32(ver, 2, 2, DEF_NO, DEF_YES, &buf[0]);          (4)
    }

…

   return (DEF_OK);                                                 (5)
}
```

L4-6(1) `HTTPs_ValReq()` processes `${variable_name}` tags from within an html file. Upon processing an HTTP request, the μC/HTTP server will call this function in order to replace a `${variable_name}` tag with an ASCII string equivalent of the requested variable's value. For example, if the web server found the string "`${OS_VERSION}`" within an html document, then this function return the string "2.81".

L4-6(2) Compare the discovered tag from within the HTML file to the text "`OS_VERSION`". If they match, process, format and return the string equivalent of the data stored within the variable "`OS_VERSION`".

L4-6(3) "`OS_VERSION`" is stored as 281, so we divide by 100 to format it properly.

L4-6(4) Convert the number 2.81 into its string equivalent ASCII string "2.81" and store it in a return buffer so that the μC/HTTP server can swap the `${OS_VERSION}` tag with the ASCII equivalent of the variable's data.

L4-6(5) Return `DEF_OK` which is an error-free return value.

Listing 4-7, HTTPs_ValRx ()

```
CPU_BOOLEAN  HTTPs_ValRx (CPU_CHAR *Variable, CPU_CHAR *Val) (1)
{
    return (DEF_OK);                                            (2)
}
```

L4-7(1) `HTTPs_ValRx()` processes variable names and values that are received as a result of a μC/HTTP form submission. It is here that a user application would process these submitted variables and perform some action based on their values. In this case, received variables and their values are ignored.

L4-7(2) Since this example does not implement HTTP forms, and no data will be posted to the web server, the example simple returns `DEF_OK` which is an error-free return value.

Listing 4-8, AppTerminalRx ()

```
#if OS_VIEW_MODULE > 0
static  void  AppTerminalRx (INT8U rx_data)                (1)
{
}
#endif
```

L4-8(1) `AppTerminalRx()` is the μC/OS-View terminal callback function. If a character is entered into the μC/OS-View terminal window, then it can be received here and processed. It is up to the user application to act accordingly if a character is received. In this case, received characters are ignored.

4.02 Example Code, app_cfg.h

This file is used to configure:

- the μC module enables
- the μC/OS-II task priorities of each of the tasks in your application
- the stack size for each task
- μC/OS-View
- μC/HTTPs
- μC/TFTP

The reason this is done here is to make it easier to configure your application from a single file.

4.03 Example Code, includes.h

`includes.h` is a 'master' header file that contains `#include` directives to include other header files. This is done to make the code cleaner to read and easier to maintain.

4.04 Example Code, os_cfg.h

This file is used to configure μC/OS-II and defines the maximum number of tasks that your application can have, which services will be enabled (semaphores, mailboxes, queues, etc.), the size of the idle and statistic task and more. In all, there are about `60` or so `#define` that you can set in this file. Each entry is commented and additional information about the purpose of each `#define` can be found in the μC/OS-II book. `os_cfg.h` assumes you have μC/OS-II V2.81 or higher but also works with previous versions of μC/OS-II.

4.05 Ex1-OS-View-TCPIP-FS-HTTPs-TFTPs-FLASH.*

These files are IAR embedded workbench project files.

5.00 Board Support Package (BSP)

BSP stands for Board Support Package and provides functions to encapsulate common I/O access functions in order to make it easier for you to port your application code. In fact, you should be able to create other applications using the AT91SAM7X-EK Evaluation Board and reuse these functions thus saving you a lot of time.

The BSP performs the following functions:

- Determine the AT91SAM7X256's CPU clock and peripheral frequencies
- Initialize the interrupt vectors if running the application from RAM
- Configure the LED I/Os
- Provide LED support functions
- Handle IRQ and FIQ ISRs
- Handling of μC/OS-II's tick timer
- μC/OS-View timer functions

5.01 Board Support Package, bsp*.*

We will not be discussing every aspect of the BSP but only cover topics that require special attention.

Your application code must call `BSP_Init()` to initialize the BSP. `BSP_Init()` in turn calls other functions as needed.

Listing 5-1, BSP_Init()

```
void  BSP_Init (void)
{
    AT91C_BASE_WDTC->WDTC_WDMR = AT91C_WDTC_WDDIS;                (1)

    LED_Init();                                                   (2)
    PLL_Init();                                                   (3)
    BSP_IntCtrlInit
    Tmr_TickInit();                                               (4)
}
```

L5-1(1) `LED_Init()` initializes the I/O pins associated with the onboard LEDs for output mode and configures all LEDs to the off state.

L5-1(2) This function reconfigures the on-chip PLL to run at 48 MHZ.

L5-1(3) This function initializes the interrupt vector table with all Dummy ISR's handler addresses. When a particular vector is required by the application, the value stored at the desired table index should be changed to that of the desired ISR handler.

L5-1(4) We then call `Tmr_TickInit()` which will initialize the PIT (Periodic Interval Timer) to generate interrupts for the μC/OS-II clock tick. The code for this function is described below.

Listing 5-2, Tmr_TickInit()

```
static  void  Tmr_TickInit (void)
{
    INT32U  counts;
    INT32U  cpu_frq;


    cpu_frq                     = BSP_CPU_ClkFreq();                                    (1)
    counts                      = ((cpu_frq * 1000) / 16 / OS_TICKS_PER_SEC) - 1;       (2)

    AT91C_BASE_AIC->AIC_SVR[AT91C_ID_SYS] = (INT32U)Tmr_TickISR_Handler;                (3)
    AT91C_BASE_AIC->AIC_SMR[AT91C_ID_SYS] = AT91C_AIC_SRCTYPE_INT_HIGH_LEVEL
                                          | AT91C_AIC_PRIOR_LOWEST;
    AT91C_BASE_AIC->AIC_ICCR    = 1 << AT91C_ID_SYS;
    AT91C_BASE_AIC->AIC_IECR    = 1 << AT91C_ID_SYS;

    AT91C_BASE_PITC->PITC_PIMR  = AT91C_PITC_PITEN | AT91C_PITC_PITIEN | counts;        (4)
}
```

L5-2(1) Read the PLL configuration registers to determine the master clock frequency. If the PLL is not enabled, this function returns the oscillator speed defined as 18,432 KHZ.

L5-2(2) Convert the system master clock frequency from KHZ to HZ by multiplying by 1000. It is then divided by 16 which is the PIT prescaler, and divided again by the number of desired ticks per second. This in turn leaves us with the value that the PIT should count up to before it resets back to 0. After 1 second, the PIT will have reset the desired `OS_TICKS_PER_SEC` number of times. We subtract 1 since the count of the PIT count starts from 0 and not 1.

L5-2(3) We configure the PIT interrupt and set its corresponding interrupt handler.

L5-2(4) Enable the PIT and specify the number of counts before an interrupt should occur.

Listing 5-3, Tmr_TickISR_Handler()

```
void  Tmr_TickISR_Handler (void)
{
    volatile  INT32U  status;


    status                   = AT91C_BASE_PITC->PITC_PIVR;            (1)
    AT91C_BASE_AIC->AIC_ICCR = 1 << AT91C_ID_SYS;                     (2)
    OSTimeTick();                                                     (3)
    AT91C_BASE_AIC->AIC_EOICR = 0;                                    (4)
}
```

L5-3(1) Acknowledge the PIT interrupt.

L5-3(2) Clear the AIC interrupt source.

L5-3(3) `OSTimeTick()` is called to handle the µC/OS-II clock tick.

L5-3(4) Issue an AIC End of Interrupt Command.

Note: When the PIT issues an interrupt, the processor vectors to `OS_CPU_IRQ_ISR()` which then calls `OS_CPU_IRQ_ISR_Handler()` (see `BSP.C`). `OS_CPU_IRQ_ISR_Handler()` and finally calls the correct handler function. In our case, this is `Tmr_TickISR_Handler()` as shown above in Listing 5-3. If `OS_TICKS_PER_SEC` is defined as `100`, as in `os_cfg.h`, then a tick interrupt occurs every 10ms.

You should note that ALL of your ISRs should be written as '`void MyISR(void)`' functions as shown. Refer to **AN-1014** for details.

5.02 Board Support Package, net_bsp.c

The source code located within `net_bsp.c` is mainly responsible for configuring the hardware pins connecting the AT91SAM7X256 and the on board Davicom PHY. However, as you will see, other small utility functions are provided as well.

Listing 5-4, NetBSP_Phy_HW_Init()

```
void NetBSP_Phy_HW_Init (void)
{
    CPU_INT32U pins;

    AT91C_BASE_PMC->PMC_PCER                = (1 << AT91C_ID_PIOB);                (1)
    AT91C_BASE_PIOB->PIO_PPUDR              = 1 << 15;                             (2)

#ifndef RMII
    AT91C_BASE_PIOB->PIO_PPUDR              = 1 << 16;                             (3)
#endif

    AT91C_BASE_PIOB->PIO_PER                = 1 << 18;                             (4)
    AT91C_BASE_PIOB->PIO_OER                = 1 << 18;                             (5)
    AT91C_BASE_PIOB->PIO_CODR               = 1 << 18;                             (6)
    AT91C_BASE_RSTC->RSTC_RMR = 0xA5000000 | AT91C_RSTC_ERSTL & (1 << 8);          (7)
    AT91C_BASE_RSTC->RSTC_RCR = 0xA5000000 | AT91C_RSTC_EXTRST;                    (8)

    while ((AT91C_BASE_RSTC->RSTC_RSR & AT91C_RSTC_NRSTL) == 0) {                  (9)
        ;
    }

    pins                      = ((CPU_INT32U) AT91C_PB2_ETX0)         |            (10)
                                ((CPU_INT32U) AT91C_PB12_ETXER)       |
                                ((CPU_INT32U) AT91C_PB16_ECOL)        |
                                ((CPU_INT32U) AT91C_PB11_ETX3)        |
                                ((CPU_INT32U) AT91C_PB6_ERX1)         |
                                ((CPU_INT32U) AT91C_PB15_ERXDV)       |
                                ((CPU_INT32U) AT91C_PB13_ERX2)        |
                                ((CPU_INT32U) AT91C_PB3_ETX1)         |
                                ((CPU_INT32U) AT91C_PB8_EMDC)         |
                                ((CPU_INT32U) AT91C_PB5_ERX0)         |
                                ((CPU_INT32U) AT91C_PB14_ERX3)        |
                                ((CPU_INT32U) AT91C_PB4_ECRS_ECRSDV)  |
                                ((CPU_INT32U) AT91C_PB1_ETXEN)        |
                                ((CPU_INT32U) AT91C_PB10_ETX2)        |
                                ((CPU_INT32U) AT91C_PB0_ETXCK_EREFCK) |
                                ((CPU_INT32U) AT91C_PB9_EMDIO)        |
                                ((CPU_INT32U) AT91C_PB7_ERXER)        |
                                ((CPU_INT32U) AT91C_PB17_ERXCK);

    AT91C_BASE_PIOB->PIO_ASR  =   pins;                                            (11)
    AT91C_BASE_PIOB->PIO_BSR  =      0;                                            (12)
    AT91C_BASE_PIOB->PIO_PDR  =   pins;                                            (13)
}
```

L5-4(1) Ensure that the peripheral clock for PIOB is enabled.

L5-4(2) Disable RXDV pull-up (since PHY has internal pull-up), and enter PHY normal mode.

L5-4(3) Put the PHY into MII mode.

L5-4(4) Enable pin 18 as GPIO controlled.

L5-4(5) Set pin 18 (PWRDWN) as an output pin.

L5-4(6) Clear the PWRDWN pin, which powers UP the PHY.

L5-4(7) Set the amount of time a PHY HW reset should be asserted for.

L5-4(8) Toggle the NRST PHY reset pin.

L5-4(9) Wait for HW reset to complete. This loop always terminates since its timer based.

L5-4(10) Set a variable containing a value for the used PHY pins.

L5-4(11) Select peripheral A use of the associated pins.

L5-4(12) Select peripheral B, and set it so no peripheral B pins are used.

L5-4(13) Set peripheral control of the associated pins.

Listing 5-5, NETBSP_EMAC_Settings_Update()

```
void NetBSP_EMAC_Settings_Update (CPU_INT32U link_speed, CPU_INT32U link_duplex)
{
    INT32U reg_val;

                                                                          (1)
    reg_val         =   AT91C_BASE_EMAC->EMAC_NCFGR & ~(AT91C_EMAC_SPD | AT91C_EMAC_FD);

    if (link_speed ==  EMAC_SPD_100) {                                    (2)
        reg_val    |=   AT91C_EMAC_SPD;
    }

    if (link_duplex == EMAC_DUPLEX_FULL) {                                (3)
        reg_val    |=   AT91C_EMAC_FD;
    }

    AT91C_BASE_EMAC->EMAC_NCFGR = reg_val;                                (4)
}
```

Since the AT91SAM7X256 has a built in EMAC and an external PHY, it is important to ensure that both the EMAC and the PHY are aware of the current link status for timing purposes. This function is called during EMAC initialization and should also be called during a PHY ISR for any type of link state change. Both link speed and duplex must be determined by querying the PHY prior to calling this function.

L5-5(1) Get current EMAC configuration and clear speed & duplex bits.

L5-5(2) If the current PHY link speed is 100, set the 100mbps bit in `reg_val`.

L5-5(3) If the current PHY duplex is set to full, set the full duplex bit in `reg_val`.

L5-5(4) Write the value of `reg_val` into the EMAC Network Configuration register.

Listing 5-6, DM9161AE_DlyAutoNegAck ()

```
void  DM9161AE_DlyAutoNegAck (void)
{
    OSTimeDlyHMSM(0, 0, 1, 0);
}
```

`DM9161AE_DlyAutoNegAck()` is called during PHY initialization from `net_phy.c` It is responsible for creating a short delay of 1 second or more after initiating auto-negotiation. After this delay has expired, further initialization of the NIC may continue. This is a user specified function which must be implemented. In this case, we use the built in `OSTimeDlyHMSM()` function of μC/OS-II, however, a delay created by any means is acceptable.

Listing 5-7, NetBSP_NIC_PhyRdWrDly ()

```
void  NetBSP_NIC_PhyRdWrDly (void)
{
    OSTimeDlyHMSM(0, 0, 0, 1);
}
```

`NetBSP_NIC_PhyRdWrDly()`is called by `NetNIC_PhyRegRd()` and `NetNIC_PhyRegWr()` every time a PHY register needs to be read or written. This function creates a delay of 1ms such that there is enough time for the register read or write to complete. The calling function uses this as way of determining whether a read or write failed due to a timeout. This is a user specified function which must be implemented. In this case, we use the built in `OSTimeDlyHMSM()` function of μC/OS-II, however, a delay created by any means is acceptable.

Listing 5-8, Time Stamp Functions

```
NET_TS                NetUtil_TS_Get            (void);
void                  NetTCP_InitTxSeqNbr       (void);
NET_TCP_TX_RTT_TS_MS  NetTCP_TxRTT_GetTS        (void);
NET_TCP_TX_RTT_TS_MS  NetTCP_TxConnRTT_GetTS_ms (void);
```

The above functions are used for initializing μC/TCP-IP sequence numbers and time stamps. They are also used for getting time stamp values that are used within various μC/TCP-IP services. These functions are user defined and must be implemented in the `net_bsp`. For a full explanation of the above functions, please see the μC/TCP-IP manual.

6.00 Demo Application

This demonstration application blinks the onboard LEDs in a pre-determined pattern and initializes μC/HTTPs, μC/TFTPs, and μC/FS for the user to interact with.

6.01 HTTP Server demo

The `index.htm` file included with this demonstration, uses the μC/HTTPs scripting capabilities. We have implemented read requests only. The demo application implements a callback function that allows the HTML page to retrieve certain data from the application. Before you can load the HTTP demo page, you must first ensure that the included `index.html` file is present on an SD/MMC card and inserted into the SD/MMC card slot ***before*** powering up the AT91SAM7X-EK. Once the web page is in place, you may view it by navigating to the internal web server using the devices IP address. In our case: `http://192.168.0.60`. If you do not have an MMC/SD card slot reader, continue to section 6.02 for instructions on how to use the TFTP Server to read and write files to the SD/MMC card over the network while it is attached to the target.

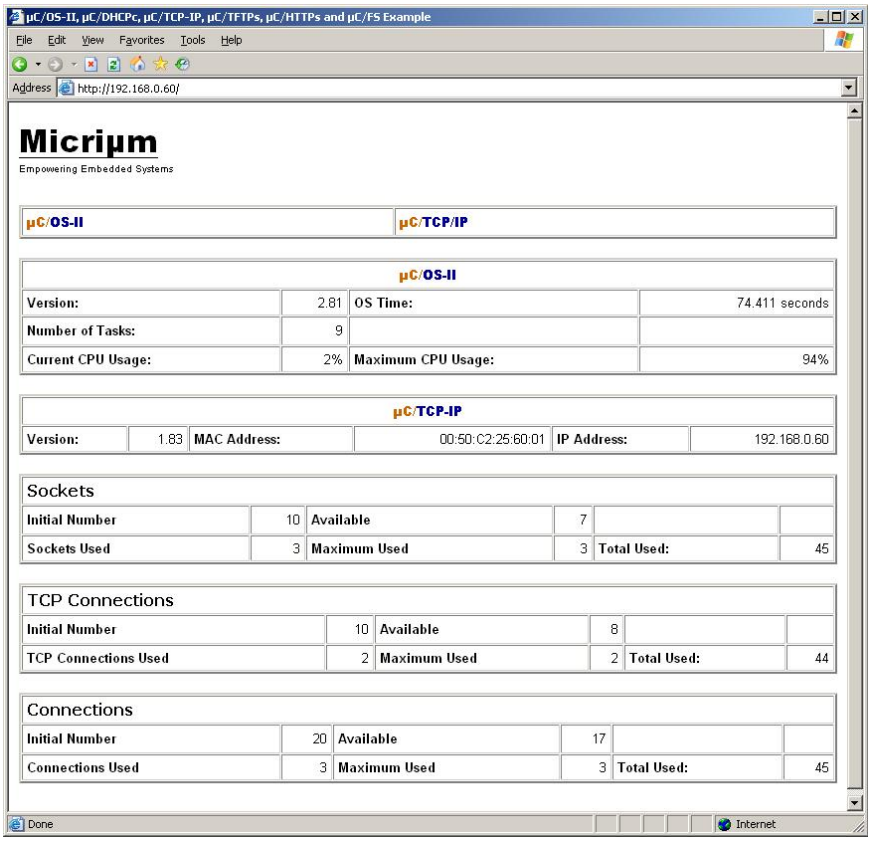

μC/OS-II, μC/DHCPc, μC/TCP-IP, μC/TFTPs, μC/HTTPs and μC/FS Example
File Edit View Favorites Tools Help
Address http://192.168.0.60/
Micriμm
Empowering Embedded Systems
μC/OS-II
μC/TCP/IP
μC/OS-II
Version: 2.81 OS Time: 74.411 seconds
Number of Tasks: 9
Current CPU Usage: 2% Maximum CPU Usage: 94%
μC/TCP-IP
Version: 1.83 MAC Address: 00:50:C2:25:60:01 IP Address: 192.168.0.60
Sockets
Initial Number 10 Available 7
Sockets Used 3 Maximum Used 3 Total Used: 45
TCP Connections
Initial Number 10 Available 8
TCP Connections Used 2 Maximum Used 2 Total Used: 44
Connections
Initial Number 20 Available 17
Connections Used 3 Maximum Used 3 Total Used: 45
Done
Internet

Figure 6-1, Browsing index.htm on the target

The `index.htm` file uses the μC/HTTPs `${TEXT_STRING}` syntax to pass information from the application to the web page. An application specific function called `HTTPs_ValRx()`parses the

syntax and build a replacement text string that the server will give to the client in place of the `${TEXT_STRING}` text. The `${TEXT_STRING}`used in this applications are:

${TEXT_STRING}	**Definition**
μC/OS-II	
${OS_VERSION}	Version number of μC/OS-II running in the application
${OS_TIME}	Current time in seconds since the application initialization
${OS_NBR_TASKS}	Number of tasks presently being scheduled
${OS_CPU_USAGE}	Current CPU usage in percent
${OS_CPU_USAGE_MAX}	Maximum CPU usage since the application initialization
μC/TCP-IP	
${NET_VERSION}	μC/TCP-IP Version number running in the application
${MAC}	The Ethernet MAC address of the Ethernet controller
${IP}	The IP address assigned to the AT91SAM7x256 via DHCP
Sockets	
${NBR_SOCK}	Number of sockets configured/reserved by the application
${NBR_SOCK_AVAIL}	Number of sockets currently available
${NBR_SOCK_USED}	Number of sockets currently used
${NBR_SOCK_USED_MAX}	Maximum number of sockets used since the application initialization
${NBR_SOCK_USED_TOT}	Total number of sockets (Open and Closed) since the application initialization
TCP Connections	
${NBR_TCP_CONN}	Number of TCP connections configured/reserved by the application
${NBR_TCP_CONN_AVAIL}	Number of TCP connections currently available
${NBR_TCP_CONN_USED}	Number of TCP connections currently used
${NBR_TCP_CONN_USED_MAX}	Maximum number of TCP connections used since the application initialization
${NBR_TCP_CONN_USED_TOT}	Total number of TCP connections (Open and Closed) since the application initialization
Connections	
${NBR_CONN}	Number of Connections configured/reserved by the application
${NBR_CONN_AVAIL}	Number of Connections currently available
${NBR_CONN_USED}	Number of Connections currently used
${NBR_CONN_USED_MAX}	Maximum number of Connections used since the application initialization
${NBR_CONN_USED_TOT}	Total number of Connections (Open and Closed) since the application initialization

You can replace `index.htm` by your own version, as long as you don't want to get data from the target itself. In this case, you would have to modify the application source code to take into considerations the new data you want to transfer via http.

6.02 TFTP Server

This demo application includes initialization for µC/TFTPs, the Micrium embedded TFTP Server. In order to transfer files to and from the embedded target, you must first have a TFTP client application. There are many free applications available from `download.com` and, there is also a DOS based client built into Microsoft Windows 2000, and XP.

The syntax for the DOS based version is as follows:

```
tftp -i <target IP> put <file_name>
```

For example: `tftp -i 192.168.0.60 put index.html` will take a file in the current working directory and transfer it to 192.168.0.60. Similarly, instead of `put`, `get` may be used in order to retrieve a file from the embedded target and store it in the current working directory.

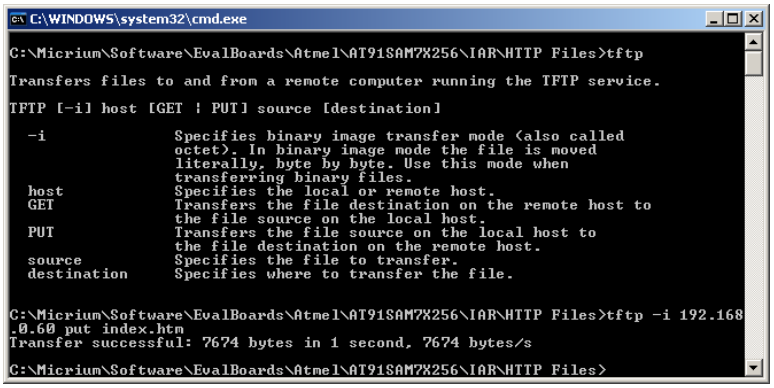

C:\WINDOWS\system32\cmd.exe
C:\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\HTTP Files>tftp
Transfers files to and from a remote computer running the TFTP service.
TFTP [-i] host [GET ! PUT] source [destination]
-i Specifies binary image transfer mode (also called octet). In binary image mode the file is moved literally, byte by byte. Use this mode when transferring binary files.
host Specifies the local or remote host.
GET Transfers the file destination on the remote host to the file source on the local host.
PUT Transfers the file source on the local host to the file destination on the remote host.
source Specifies the file to transfer.
destination Specifies where to transfer the file.
C:\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\HTTP Files>tftp -i 192.168.0.60 put index.htm
Transfer successful: 7674 bytes in 1 second, 7674 bytes/s
C:\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR\HTTP Files>

Figure 6-2, Transmitting index.htm via TFTP

Ping

In order to troubleshoot and or test network connectivity, µC/TCP-IP provides ICMP Ping functionality. You may ping the embedded target by opening up a command window, or using any graphical utility of your choice. Below is a demonstration of the windows command prompt ping utility as it applies to this example.

Example: `<command prompt>ping 192.168.0.60`, return. `<command prompt>` may be any text pertaining to the starting location of the command prompt tool on your computer. To bring up a command window under Microsoft Windows 2000 or XP, simply click start, run, and type `cmd` and press return. For an earlier version of Windows, type `command` and press return. Then enter the ping command above and press return. You should receive reply from messages. If you receive request timed out messages, ensure that you are pinging the proper IP address and that the network cable is properly connected. Restart the embedded hardware by means of power cycling, not the onboard reset switch, and try again. You may also use the debugger to restart the application if desired.

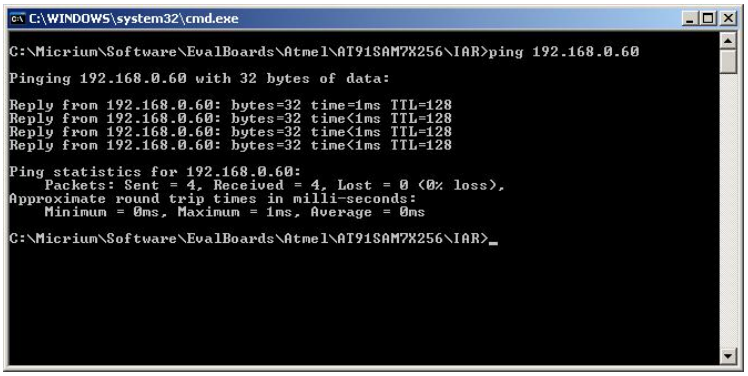

C:\WINDOWS\system32\cmd.exe
C:\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR>ping 192.168.0.60
Pinging 192.168.0.60 with 32 bytes of data:
Reply from 192.168.0.60: bytes=32 time=1ms TTL=128
Reply from 192.168.0.60: bytes=32 time<1ms TTL=128
Reply from 192.168.0.60: bytes=32 time<1ms TTL=128
Reply from 192.168.0.60: bytes=32 time<1ms TTL=128
Ping statistics for 192.168.0.60:
Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
Minimum = 0ms, Maximum = 1ms, Average = 0ms
C:\Micrium\Software\EvalBoards\Atmel\AT91SAM7X256\IAR>

Figure 6-3, Pinging the Embedded Target via the Ping command

References

μC/OS-II, ***The Real-Time Kernel, 2nd Edition***
Jean J. Labrosse
R&D Technical Books, 2002
ISBN 1-57820-103-9

Embedded Systems Building Blocks
Jean J. Labrosse
R&D Technical Books, 2000
ISBN 0-87930-604-1

Contacts

IAR Systems
Century Plaza
1065 E. Hillsdale Blvd
Foster City, CA 94404
USA
+1 650 287 4250
+1 650 287 4253 (FAX)
e-mail: Info@IAR.com
WEB : www.IAR.com

Atmel Corporation
2325 Orchard Parkway
San Jose, CA 95131
USA
+1 408 441 0311
WEB: www.Atmel.com

AT91SAM7X256.pdf
AT91SAM7X-EK-RevA.pdf

CMP Books
6600 Silacci Way
Gilroy, CA 95020 USA
Phone Orders: 1-800-500-6875
1-408-848-3854
Fax Orders: 1-408-848-5784
e-mail: cmp@rushorder.com
WEB: http://www.cmpbooks.com

Micriμm
949 Crestview Circle
Weston, FL 33327
USA
+1 954 217 2036
+1 954 217 2037 (FAX)
e-mail: Jean.Labrosse@Micrium.com
WEB: www.Micrium.com